TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 1802R022 – Informatika a logistika

Aplikace pro synchronizování dokumentů v P2P síti

Synchronization of Documents in Peer-To-Peer environment

Bakalářská práce

Autor: **Lukáš Nový**

Vedoucí bakalářské práce: Ing. Roman Špánek, Ph.D.

Konzultant: Ing. Pavel Tyl

V Liberci 21. 5. 2009

Originál zadaní práce

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé BP a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Tímto bych chtěl poděkovat Ing. Romanovi Špánkovi, Ph.D. za vedení této bakalářské práce, cenné rady a připomínky při vypracování. Dále bych chtěl poděkovat Ing. Aleně Gregové za podnětné poznámky k tématu.

Abstrakt

V úvodní části práce budou přehledně rozděleny počítačové systémy, dle použité architektury. Dále bude práce ve svém teoretickém úvodu, pojednávat o distribuovaných systémech a to především o peer to peer architektuře. Práce se bude dále zabývat přednostmi a nedostatky této architektury a to především v porovnání s architekturou Klient-server. Samostatná část pak bude věnována charakteristickým vlastnostem peer to peer architektury. V praktické části bude nejprve popsána motivace pro vznik práce. Bude nastíněno řešení daného problému a následně bude popsána funkčnost vlastní aplikace. Její fyzická podoba a algoritmus, včetně popisu grafického uživatelského rozhranní.

Klíčová slova: Peer-to-peer, klient-server, synchronizování dokumentů, P2P aplikace

Abstract

Computer systems will be transparently sorted in the initiative part of the thesis. Then in its theoretical opening, the thesis will discuss distributed systems, especially their one part, peer to peer. The thesis will then deal with advantages and disadvantages of this architecture. Especially compared to architecture client-server. A separate part will be then devoted to characteristic features of peer to peer architecture. The main part of the thesis is a practical part. Initially, goals and reasons of making the thesis will be described. The solution of query will be drawn out. Subsequently, the functionality of own application will be described. Its physical form and algorithm. The description of graphical user interface will not be missing. Accomplished results will be summarized in the conclusion.

Keywords: Peer-to-peer, client-server, synchronization of documents, P2P application

Obsah:

Přehled zkratek, symbolů

P2P - Peer - to - peer

DNS – Domain name system

TTL – Time to live

UDP – User datagram protocol

WAN – Wide area network

TCP – Transmision control protocol

NAT – Network adress translation

CSV – Comma - separated values

HTTP – Hypertext transfer protocol

TFTP – Trivial file transfer protocol

FTP- File transfer protocol

Úvod

Peer to peer systémy jsou dnes neodmyslitelnou součástí informačních technologií. Proto budou přiblíženy principy jejich fungování a některé podstatné myšlenky na kterých jsou založeny. Teoretická část práce by měla poskytnout, ucelený pohled na peer to peer systémy a důvody pro jejich zavedení. V praktické části bude řešen konkrétní problém, P2P sítí. Při kterém budou využity teoretické poznatky z úvodních částí práce. Ověření si znalostí, vytvoření funkční a prakticky využitelné aplikace, která je založena na principech P2P sítí, to jsou hlavní cíle této práce.

1 Počítačové systémy

1.1 Centralizované počítačové systémy

Počítačové systémy si můžeme obecně rozdělit do dvou skupin. Tou první jsou *centralizované počítačové systémy*. Centrální prvek zde může být zastoupen mainframem (sálový počítač velkých rozměrů, vyznačující se vysokou spolehlivostí a výkonem, schopný běžet velmi dlouhou dobu bez přerušení a schopný zpracovávat velké množství dat). Centralizovaný počítačový systém tedy reprezentuje jediná pracovní jednotka, kterou představuje buď jedno, nebo víceprocesorové zařízení, které dosahuje vysokého výpočetního výkonu, můžeme si ho tedy definovat jako superpočítač. Všechny aplikace, data, služby a procesy jsou uloženy a běží jen na tomto superpočítači. Centralizované počítačové systémy se vyznačují tím, že klient není fyzicky reprezentován žádným procesorem, pamětí či pevným diskem. Všechny tyto prostředky jsou sdíleny superpočítačem. Klient je reprezentován pouze vstupy a výstupy, jako je klávesnice nebo monitor.

1.2 Distribuované počítačové systémy

Druhou skupinu tvoří takzvané *distribuované počítačové systémy*. V distribuovaném počítačovém systému jsou různé komponenty, prvky a aplikace situované v různých počítačích a ty jsou spojeny sítí. Klienti mají své vlastní procesory, paměti a další komponenty. Části jednotlivých programů mohou běžet současně na dvou či více počítačích v síti. Tyto počítače potom mezi sebou komunikují a koordinují svou činnost výhradně metodou zasílání zpráv. Právě distribuovanými počítačovými systémy se budeme zabývat dále v této práci.

2 Klient-server vs. Peer to peer

2.1 Klient-server architektura

2.1.1 Základní pojmy

Distribuované počítačové systémy tvoří dvě velice rozdílné architektury. Tou první je velmi dobře známá a rozšířená architektura *Klient–server*.

Nejdříve, ale několik zásadních pojmů, které se týkají nejen této architektury. *Server* si můžeme představit jako program nebo také počítač, na kterém běží programy a aplikace, server poskytuje klientům v síti různé služby dle požadavku klienta. Jedná se například o přidělení diskového prostoru, sdílení dat a programů. Server se dá definovat jako obslužná stanice zajišťující chod sítě. Servery mohou být různých druhů :

- Souborový
- Databázový
- Tiskový
- DNS a mnoho dalších.

Klient je počítač(uzel) nebo přesněji softwarový program, který vznáší požadavky vůči serveru. Uživatel prostřednictvím klienta požaduje od serveru přístup k tiskárnám, datům a různým souborům. Uživatel služby využívá tak, jako kdyby byly přístupné přímo lokálně na jeho stanici.

2.1.2 Přednosti a nedostatky

Klient–server architektura představuje model, v němž jsou vzájemně klienti a servery spojeni v síti. Typicky se jedná o jeden nebo několik serverů na mnohonásobně vyšší počet stanic. Klient–server architekturu si můžeme dále rozdělit na *přímou a hierarchickou*. O přímý Klient–server model se tedy jedná v případě, že všichni klienti komunikují pouze s jediným serverem. Jedná se o *přímou komunikaci* s jedním serverem. Druhým typem je *hierarchická architektura*, která má oproti přímé výhodu v možnosti dalšího rozšiřovaní sítě. V hierarchickém modelu jsou servery z nižší úrovně v roli klienta vůči serveru z úrovně vyšší. Příkladem hierarchického modelu může být implementace DNS serveru, kde je právě využito několik úrovní serverů, které jsou si navzájem nadřazeny. Při implementaci Klient-server architektury je nutností použití vysoce spolehlivého centrálního prvku. Klient–server architektura nám poskytuje

vysokou kontrolu nad děním v síti, to znamená identifikaci uživatelů, určitou kontrolu jejich činnosti, případně omezení některých nežádoucích klientských práv.

Tato architektura má však také mnoho nedostatků. Mezi jeden z největších patří obtížná rozšiřitelnost, která je dána především kapacitou serverů a jejich fyzickou polohou. Dále jindy oceňovaná přítomnost centrálního prvku, může mít v případě jeho výpadku či poruchy katastrofální dopad na klienty. V případě výpadku serveru je tedy mimo provoz celá síť, což uživatelům znemožní práci se síťovými prostředky, v krajních případech znemožní práci vůbec.

Nejhorším případem je pak v případě špatného systému zálohování ztráta všech dat, která jsou na serveru uložena. Klient-server architektura je také velice nákladná na implementaci. Vezměme v potaz vysoké náklady na pořízení samotného serveru, softwarový systém nutný ke správě sítě a zařízení fyzické vrstvy. Server je také poměrně náročný na administraci. Klient-server architektura tedy vyžaduje přítomnost jednoho či více administrátorů, kteří systém řídí, spravují a udržují v chodu. Klient-server architektura se osvědčila především u velmi úspěšných modelů jako je http, ftp či tftp.

2.2 Peer to peer architektura

2.2.1 Základní pojmy

Druhou architekturou, která se začíná stále více dostávat do povědomí uživatelů a snaží se řešit problémy a nedostatky spojené s Klient–server architekturou, je *Peer to Peer* architektura (dále *P2P*), neboli architektura “rovný s rovným”.

Původní myšlenkou při tvorbě základů této architektury bylo, že svět bude spojen a bude eliminován nežádoucí vliv centralizace a systému, který je nutné administrátorsky spravovat. Sdílení počítačových zdrojů a služeb, přímou výměnou mezi systémy, takto například definovala P2P systémy firma Intel [2]. Tyto zdroje a služby se dají popsat jako výměna informací, diskového prostoru a podobně. To znamená, že uzel poskytuje ostatním uzlům své služby a sám si může brát od ostatních. Každý uzel vystupuje zároveň v roli serveru i klienta. Jakýkoliv uzel je schopen zahájit komunikaci. Žádné centralizované zdroje dat a služeb. P2P poskytuje určitou formu demokracie, zároveň však představuje hrozbu z pohledu právní ochrany - nelegálního stahování hudby, filmů, knih a mnoha jiných věcí.

V P2P modelu tedy klient zároveň plní i roli serveru. Uzly by měly být zcela nezávislé, a pokud možno nepodléhat žádné administraci. P2P síť je plně dynamická, uzly přistupují a opouštějí síť dle své potřeby. Uzly mají měnící se možnosti a schopnosti. Toto je tedy stručná úvodní definice architektury, jejímž studiem se zabýváme v této práci.

2.2.2 Typy Peer to peer sítí

P2P sítě mohou mít různou podobu své implementace. Jednou z nich je takzvaná P2P architektura *hybridní*. Ačkoliv se tedy jedná o P2P model, některé úlohy jsou paradoxně centralizované. Tato hybridní P2P síť funguje tak, že existuje centrální prvek, tedy server. Ten však neplní úlohu poskytovatele požadovaných souborů, je pouze jakýmsi prostředníkem pro budoucí přenos dat. Klient připojený do takovéto hybridní sítě tedy nejprve musí nahrát svůj seznam souborů na centrální prvek sítě. Centrální prvek uchovává pouze informace o sdílených datech, IP adrese, případně typu připojení jednotlivých klientů. Pokud tedy nějaký klient zahájí komunikaci, pošle nejprve svůj dotaz k centrálnímu prvku. Pokud centrální prvek nalezne v seznamu souborů hledaný dotaz, odpoví klientovi, který vznesl požadavek. Odpověď obsahuje mimo jiné IP adresu cílového uživatele. Uživatelé se následně spojí a je možné vyměňovat data či sdílet služby, bez účasti centrálního prvku. Hybridní P2P model tedy nenabízí v plné míře výhody které z definice P2P architektury plynou. Centrální server má opět určitou kapacitu a při jeho výpadku je vyhledávání dat či služeb nemožné.

Můžeme si tento model tedy shrnout do dvou bodů:

- Hledání probíhá centralizovaně
- Samotný přenos je již plně P2P

Známé aplikace, které využívaly tento model, jsou například Napster, Groove, Magi [2].

Druhým typem P2P architektury je takzvaná *čistá* (pure P2P) *architektura*. Jedná se již plně o komunikaci bez účasti centrálního prvku v síti. Tuto takzvanou čistou P2P architekturu lze dělit jako:

- Strukturovanou
- Nestrukturovanou.

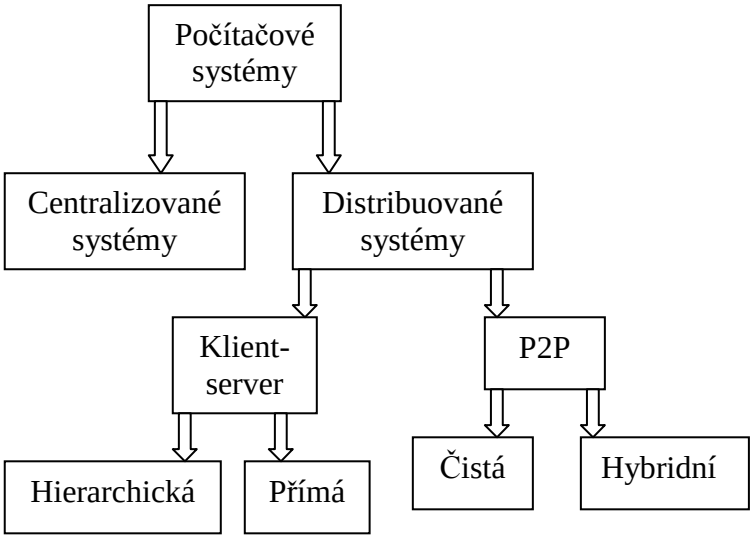
2.2.3 Algoritmy

V čisté nestrukturované P2P architektuře se používá několik vyhledávacích algoritmů. Jako nejzákladnější se dá brát *jednoduchý tok*, kdy zašle uzel vznášející požadavek zprávu všem svým sousedním uzlům. V případě, že některý z těchto sousedních uzlů má odpovídající výsledek, který požadoval uzel, jenž požadavek vznesl, je iniciátor komunikace informován a může přijmout data přímo od tohoto uzlu. V opačném případě, sousední uzel sníží hodnotu *TTL* (Time to live je parametr obsahující hodnotu, uloženou v hlavičce paketu. Určuje kolika směrovači, v našem případě uzly, může paket projít. Obsahuje číslo, které se při přechodu přes jednotlivé části sítě snižuje o 1. Pokud se číslo sníží až na 0, paket již nebude dál sítí šířen a zanikne.), a pošle požadavek dalším uzlům, které s ním sousedí. Tento proces se opakuje tak dlouho, dokud není nalezen odpovídající výsledek nebo dokud hodnota pole TTL není rovna 0.

Další metodou je takzvané *opakované zvyšování*. To znamená, že hledání začíná s malou hodnotou pole TTL. V případě že není nalezen výsledek, do vypršení hodnoty TTL, začne nové hledání z vyšší hodnotou TTL. Opět se hledání opakuje, dokud není nalezen odpovídající výsledek nebo pokud není překročena hodnota TTL. U této metody je využito pole, v němž sou pro každé hledání nastaveny hodnoty TTL, například TTL (1, 3, 5). První hledání potom proběhne s hodnotou TTL 1. V případě neúspěchu bude mít nové hledání hodnotu TTL 3 a tak dále.

Následující metoda se dá popsat jako *hledání naslepo*. Uzel vznese požadavek, ale vybere si pouze jediného souseda pro zaslání požadavku. Výběr uzlu, kterému je zaslán požadavek, podléhá určitému kritériu. Pokud je uzel spolehlivý, to znamená, že ze všech sousedních uzlů nejčastěji odpovídá kladně na požadavky, bude vybrán jako první. V případě, že není požadovaný výsledek nalezen, je vybrán uzel, který je ve spolehlivosti na druhé pozici.

Poslední metodou je hledání na základě dříve *získaných informací*. Při použití tohoto algoritmu uchovává každý uzel ve své cache paměti umístění souborů, které byly již vyhledávány při příjmu předchozích požadavků. V případě, že uzel nalezne ve svém úložišti umístění požadovaného souboru, kontaktuje přímo držitele tohoto souboru a soubor dostane. Jakmile je soubor nalezen, je za pomocí zpětné cesty dotazů o umístění souboru informován uzel, který vznesl požadavek. Příkladem aplikace, která využívá přímou nestrukturovanou P2P architekturu, je Gnutella[1].



Obrázek 1 Dělení počítačových systémů

2.2.4 Strukturované Peer to peer

Druhou generací čistého P2P modelu se dá nazvat *strukturovaná architektura*

. Tato architektura se vyznačuje:

- vlastní organizací
- rovnoměrným rozdělení výkonu
- velice nízkou chybovostí

Hlavním rozdílem oproti nestrukturovaným P2P je garance určitého počtu přeposlání požadavků pro doručení odpovědi. Základem strukturovaných P2P je hash tabulka DHT (distributed hash table). Do tabulky jsou ukládána párová data, prvním z nich je klíč a druhým je hodnota. Klíč si můžeme představit jako název souboru, hodnotu pak jako jeho obsah. Každý uzel tedy uchovává v systému pomocí souboru DHT párové informace obsahující klíč a hodnotu. Každému uzlu je navíc přiřazen unikátní ID identifikátor, což je hodnota podobná IP adrese. Podstatou této komunikace je nalézt uzel, který se skrývá za hledaným klíčem. Aby byl uzel schopen klíč nalézt, existuje takzvaná mapa klíčů k uzlům. Klíče jsou mapovány stejnoměrně ke všem uzlům v síti. Každý uzel udržuje informace jen o několika ostatních uzlech.

Strukturované P2P mělo za cíl zefektivnit vkládání, vyhledávání a mazání klíčů a hodnot. Stejně tak směrování požadavků k uzlům. To znamená, že není potřeba zasílat velké množství zpráv, jako tomu bylo při vyhledávání v nestrukturovaných P2P. Na

základě DHT je postaveno mnoho dalších protokolů strukturovaných P2P, jako jsou například Chord, Pastry nebo CAN.

Na závěr této části práce shrneme některé důležité pojmy, které z práce plynou. P2P můžeme brát v potaz jako alternativu ke Klient–server architektuře. Zatímco Klient–server architektura je pevně řízena, P2P má vlastní organizaci. Klient-server má nějakou existující infrastrukturu, zatímco P2P mohou tvořit takzvané ad – hoc systémy [4]. Pokud jde o přístup k datům, u P2P je nutné nejdříve “objevit“ jejich držitele, u druhé architektury jde pouze o nahlédnutí na diskový prostor serveru. Klient–server architektura je statická, P2P se vyznačuje svou dynamikou. P2P dává volnost uživatelský názvům, zatímco Klient–server je založen na DNS.

3 Charakteristické rysy P2P systémů

P2P systémy mají několik typických rysů a také problémy, které je nutné řešit, aby mohlo být dosaženo jejich implementace. V následující části se budeme zabývat například decentralizací, rozšiřitelností, anonymitou, optimalizací výkonu, bezpečností a dalšími typickými vlastnostmi, kterými se vyznačuje model P2P.

3.1 Decentralizace

V modelu Klient–server jsou informace a data situovány na centrálním serveru. Dále jsou pak rozdělovány ke klientům prostřednictvím počítačové sítě. Počítačové systémy s přítomností centrálního prvku jsou ideálním řešením pro některé typy úloh a aplikací. Jako příklad si můžeme uvést bezpečnost či správu uživatelských práv, kdy se tyto prvky dají daleko snáze a jednodušeji zajistit pomocí centrálního prvku. Pomineme–li možné omezení z důvodu nízké propustnosti sítě, je i přes stále se zvyšující výkon hardwaru a jeho poměrně příznivé ceny takovéto řešení stále velmi nákladné na pořízení v porovnání s P2P systémem a také náročné na nastavení. V neposlední řadě také náročné z pohledu spravování a údržby. V plně decentralizovaném systému, jedná se tedy o takzvanou čistou P2P architekturu popsanou dříve, jsou si všichni uživatelé rovni. Z tohoto důvodu je však zavedení takovéhoto systému poměrně obtížné.

Jak již bylo řečeno, jedním z důvodů je absence centrálního prvku, který má přehled o uživatelích v síti nebo zná jejich seznam souborů. To je jeden z důvodů, proč je mnoho P2P systémů založeno na hybridní architektuře. Typickým příkladem tohoto systému je Napster. V plně decentralizovaných systémech, jakými jsou například Gnutella nebo Freenet se hledání v síti stává obtížným. Například právě při používání Gnutelly musí uživatel znát IP adresy jiných uzlů nebo používat takzvaný host list s IP adresami již známých uzlů.

3.2 Rozšiřitelnost

Decentralizace počítačových systémů přispívá velkou mírou ke zlepšení jejich rozšiřitelnosti. Rozšiřitelnost je omezena různými faktory. Jedním z nich je množství centrálních operací, které je třeba vykonat, nebo činnosti, které je třeba spravovat. Rozšiřitelnost také závisí na poměru přenosu a výpočetní síly mezi dvěma uzly v P2P systému. Aplikace, které mají tento poměr blížící se nule, jsou velmi snadno rozšiřitelné.

3.3 Anonymita

Jedním z cílů P2P systémů je umožnit uživatelům, používat P2P aplikace bez starosti o další následky spojené s jejich užíváním. Dalším předpokladem je garance toho, že nebude možná cenzura digitálního obsahu. Anonymitu si v našem případě můžeme představit tak, že autor či tvůrce nějakého dokumentu nebude moci být identifikován. Informace o osobě, která dokument šíří, nebude možné zjistit. Čtenáři nebo lidé, kteří shlédnou dokument, nemohou být identifikováni. Server, který obsahuje dokument, nemůže být odhalen na základě tohoto dokumentu. Servery neznají, jaké dokumenty jsou na ně ukládány, a konečně není možné ze serveru zjistit, co bylo odpovědí na uživatelův dotaz.

Můžeme si tedy definovat tři základní druhy anonymity:

- Anonymita odesílatele
- Anonymita příjemce
- Anonymita vzájemná, při níž je skryta identita jak odesílatele, tak příjemce, a to jak mezi nimi navzájem, tak i pro ostatní uživatele.

Mimo těchto druhů anonymity je také velmi důležité zmínit se o stupních anonymity a způsobech, jak jí lze dosáhnout. Stupeň anonymity lze rozdělit do čtyř částí

- prvním je absolutní soukromí
- druhým takzvaný stupeň mimo podezření
- třetím stupněm je takzvané možné podezření
- posledním stupněm je nechráněnost.

Například, stupeň mimo podezření znamená, že i přesto, že jakýkoliv uživatel může prohlédnout evidenci zaslaných zpráv, tak pravděpodobnost, že pravý odesílatel zprávy bude objeven, je stejná asi jako pravděpodobnost toho, že bude objeven jakýkoliv jiný uživatel. Dále si uvedeme některé způsoby, které jsou vhodné pro dosažení různých druhů a způsobů anonymity v P2P systémech.

Multicasting nebo také broadcasting je používán pro zajištění anonymity příjemce. Uživatelé, kteří chtějí zůstat v anonymitě, vytvoří takzvanou *multicastovou* skupinu. Účastník, který chce získat nějaký soubor a přitom zůstat v anonymitě, vstoupí do multicastové skupiny. Uživatel, jenž má požadovaný soubor, jej zašle skupině, v níž je žadatel. Identita žadatele je tedy efektivně skryta, nezná ji ani uživatel, který soubor zaslal, ani ostatní členové multicastové skupiny.

Navádění k nesprávné IP adrese je používáno pro zajištění anonymity odesílatele. Při použití protokolů, jako je například UDP, může být anonymita odesílatele dosažena navedením na nesprávnou IP adresu. To ale není vždy možné provést, mnoho z poskytovatelů internetového připojení používá filtraci paketů, které vzešly z nesprávných IP adres. Kromě změny adresy odesílatele, může být podobnou technikou také zajištěna anonymita změnou identit komunikujících stran.

Utajená cesta – namísto přímé komunikace, dvě zúčastněné strany komunikují přes prostřední uzel. Strana, která požaduje neodtajnit svou identitu, připraví skrytou cestu s ostatními stranami, které komunikují. To bude vypadat jako konec cesty. Různou délkou utajené cesty a změnou vybrané cesty je možné dosáhnout různých stupňů zaručené anonymity.

Metoda Aliasů – LPWA(Lucent personalized web assistent) je proxy server, který generuje shodné a tudíž nevypátratelné aliasy pro klienty serveru. Techniky tohoto druhu zajistí anonymitu odesílatele a také možnost spolehnout se na důvěryhodnost proxy serveru.

Nedobrovolné umístění je zajímavý, nový přístup, který řeší problém anonymity nedobrovolným umístěním dokumentu na hostitelském uzlu. Protože je umístění nedobrovolné, hostitel nenese odpovědnost za vlastnictví dokumentu.

3.4 Vlastní organizace

V kybernetice si můžeme pojem vlastní organizace definovat jako proces, při kterém je uspořádání systému tvořeno samovolně. To tedy znamená, že jinak by byl proces samovolného růstu kontrolován prostředím nebo například externím systémem. V P2P systémech je vlastní organizace důležitá z důvodu rozšiřitelnosti, odolnosti proti selhání a ceny na pořízení. P2P systémy mohou růst nepředvídatelně v počtu druhů systémů, počtu uživatelů a možného výkonu. V těchto směrech je velmi těžké odhadnout budoucí vývoj P2P systémů. Úroveň růstu má za následek zvýšenou

pravděpodobnost výskytu selhání, které vyžadují vlastní údržbu a opravy systému. To může vést i k dočasnému odpojení systému. Je velmi těžké udržet určitou konfiguraci systému aktuální na delší dobu. Pro přizpůsobení konfigurace je nutné mít přehled o tom, jaké dopady bude mít neustále připojování a odpojování uzlů do systému. Mít vyhrazené zařízení nebo administrátory pro správu systému by však bylo nákladné. Proto je P2P systém spravován rovnoměrně mezi uživateli.

3.5 Cena vlastnictví

Jedním z předpokladů, které by měly P2P systémy dodržet, je sdílení vlastnictví. Sdílení vlastnictví snižuje cenu systému, sdíleného obsahu a náklady na celkovou správu. Toto je základní věc, která by měla být aplikována ve všech třídách P2P systémů a distribuovaných systémů vůbec. Uveďme si příklad - P2P systém jako SETI@home se dá porovnávat s nejrychlejšími světovými superpočítači, přesto je jeho cena jen zlomkem pořizovací ceny a dalších věcí nutných k provozu takovýchto superpočítačů. V P2P spolupracujících a komunikačních systémech je absence centrálního prvku, na kterém jsou uložena data, faktorem, který přispívá ke snížení ceny vlastnictví a spravování.

Ještě se zmíníme o řešení v bezdrátové komunikaci, jedná se o takzvané parazitní sítě. K bezdrátovému přenosu dochází umožněním sdílení domácími uživateli vytvořením 802.11b pásem, mezi uživateli. Takovéto sítě, mohou tedy konkurovat bezdrátovým sítím poskytovaným společnostmi neporovnatelně menšími náklady.

3.6 Ad-Hoc

V těchto sítích se navzájem spojují klienti, kteří jsou v rovnocenné pozici P2P. Ad-hoc typ spojení má velmi silný dopad na P2P systémy. V distribuovaném světě počítačů nemohou být souběžné aplikace použity na všech systémech po celou dobu. Některé systémy budou dostupné jen po určitý čas a některé nebudou dostupné vůbec. Při použití P2P systémů a aplikací musíme brát zřetel na jejich ad-hoc charakter a být schopni ovládat systém při neustálém připojování a odebírání uživatelů ze systému. V P2P systémech a aplikacích určených ke vzájemné spolupráci je ad-hoc povaha připojení běžná. Uživatelé těchto systémů stále více používají mobilní zařízení, díky kterému mají lepší možnost připojení do internetu, a jsou tudíž častěji dostupní pro spolupráci. Mimo to ne všichni budou připojeni do internetu. K tomu jsou vhodné především ad-hoc sítě založené na 802.11b, Bluetooth nebo přenosech přes infraport.

Tyto technologie jsou však omezeny svým dosahem. Proto P2P systémy a aplikace potřebují být vytvořeny tak, aby byli schopny pracovat s náhlými odpojeními a opětovnými přístupy do ad-hoc skupiny uživatelů.

3.7 Výkon

Výkon v P2P systému je důležitou záležitostí. Z důvodu decentralizovaného charakteru tohoto modelu je výkon ovlivněn třemi typy prostředků:

- Zpracováním
- Možnostmi ukládání
- Připojením sítě

Šířka síťového pásma může hrát významnou roli ve WAN(wide area network) sítích. Šířka pásma je hlavním faktorem obzvláště tehdy, když je v síti šířeno velké množství zpráv a je přenášeno velké množství souborů mezi mnoha uzly. To limituje rozšiřitelnost systému. Při chápání smyslu slova výkon v tomto kontextu nedáváme tolik důraz na řády milisekund, ale spíše se snažíme odpovědět si na otázku, kolik času nám zabere získání souboru nebo kolik z šířky pásma nám zaberou dotazy. V centrálně uspořádaných systémech jako jsou Napster nebo Seti@home je součinnost mezi uzly zprostředkována a kontrolována centrálním serverem, ačkoli uzly se později mohou kontaktovat přímo navzájem mezi sebou. To činí tyto systémy náchylné k problémům, kterým obvykle čelí centralizované servery.

K překonání omezení, která sebou přináší centralizovaný prvek, bylo pro hybridní P2P architektury navrženo řešení na bázi DNS serveru. DNS je příkladem hierarchického P2P systému, který zlepšuje výkon definováním stromu takzvaných koordinátorů, kde je každý tento koordinátor zodpovědný za určitou skupinu uzlů. Aby mohlo být dosaženo komunikace mezi uzly v různých skupinách, komunikaci řídí koordinátor z nejvyšší úrovně. V decentralizovaných systémech jako Gnutella a Freenet tento koordinátor chybí. Uzly používají mechanismus zasílání zpráv pro vyhledání informací a dat. Problém takovýchto systémů je v zasílání velkého počtu zpráv přes mnoho skoků (skok nebo také hop je směrovač, v našem případě uživatel, přes kterého je paket (dotaz) přeposlán dále. Každý uživatel, přes kterého je dotaz přeposlán, si můžeme představit jako jeden skok. Maximální počet skoků je nastaven v poli TTL. To

se negativně projeví na šířce pásma a času, za jaký uzel obdrží odpověď na svůj dotaz. Nyní si uvedeme tři klíčové metody používané pro optimalizaci výkonu.

Replikace – při této metodě je například kopie souboru umístěna blíže k žádajícímu uzlu. Takto se minimalizuje vzdálenost mezi uzlem, který soubor požadoval, a uzlem který jej poskytuje. Geografické rozdělení uzlů pomáhá omezit zatížení na straně uzlu i sítě. V kombinaci s inteligentním směrováním, replikace pomáhají zmírnit zpoždění způsobené vzdáleností zasíláním žádostí k blíže umístěným uzlům.

Caching – redukuje délku cesty, která je nutná k dosažení souboru, a tedy počet zpráv vyměněných navzájem mezi uzly. Omezení takových přenosů je důležité, protože komunikační latence mezi uzly je vážnou překážkou pro dosažení výkonu, které musí P2P systémy čelit. Cílem cachingu je tedy minimalizovat latenci při přístupu k uzlům, maximalizovat propustnost dotazů a samozřejmě optimalizovat pracovní zatížení systému.

Inteligentní směrování a síťové uspořádání – Pro naplnění potencionálu, kterým P2P systémy disponují, je důležité pochopit a blíže prozkoumat společenské vazby mezi uživateli P2P. Princip tedy spočívá v tom, že budou vybráni uživatelé s podobnými zájmy a připojení mezi uzly bude dynamické, aby mohlo být zajištěno, že uživatelé s vysokým stupněm podobných zájmů budou připojeni blíže. Takovéto sblížení těchto uživatelů rapidně snižuje počet zaslaných zpráv v síti a počet uživatelů, kteří musí zpracovávat žádost před nalezením kladného výsledku.

3.8 Bezpečnost

V této oblasti P2P systému se vyskytuje několik možných metod zabezpečení.

Multi – key kódování – Systémy pro sdílení souborů chtějí chránit sdílené objekty stejně jako anonymitu autorů nebo vlastníků souborů. Konkrétně systém Publius používá metodu veřejných klíčů založenou na asymetrickém šifrovacím mechanismu [3].

Sandboxing – Některé distribuované systémy vyžadují stažení kódu na uživatelské zařízení. Je velmi důležité chránit uživatelská zařízení před potencionálním škodlivým kódem. Ochrana uživatelských zařízení obvykle zahrnuje zavedení bezpečnostních opatření, aby je cizí kód nemohl zničit a důvěrná data nemohla proniknout ke

škodlivým částem. Takové technice se říká Sandboxing [3]. Tato technika zajistí, že škodlivý kód bude izolován a bude mu zabráněno aby byl spuštěn.

Správa digitálních práv – V P2P systémech je kopírování velmi snadnou záležitostí. Je nezbytné zaručit proto autorům ochranu jejich duševního vlastnictví. Jeden ze způsobů, jak dosáhnout ochrany duševních práv, je přidání digitálního podpisu k souboru.

Dobrá pověst a zodpovědnost – V P2P systémech je často důležitá dobrá pověst uživatelů. V opačném případě může ztratit dobré jméno celý systém. Jako "dobrý" nebo "užitečný" je označen uživatel, který například poskytuje zajímavé soubory. Parazitem se dá nazvat uživatel, který soubory v P2P systému pouze požaduje, ale přitom sám žádné neposkytuje. Takovýto uživatel má obvykle reputaci mezi uživateli velmi nízkou. Bylo by tedy třeba vymyslet a zavést mechanismus, který by takovéto chování eliminoval. V současnosti se systémy spoléhají pouze na vzájemné hodnocení mezi uživatelskou komunitou. To však neposkytuje žádnou záruku toho, že takovéto parazitní chování vymizí.

Firewally – Hlavním rysem P2P modelu je požadavek na přímé spojení mezi uzly. Nicméně, v prostředí interních sítí jsou uzly odděleny od externích, tedy typicky od internetu. Tímto je ale omezen přístup k některým aplikacím z důvodu, že mnoho firewallů blokuje příchozí TCP spojení. To znamená, že zařízení za firewallem nejsou dostupná ze zařízení z externích sítí. Podobný problém mají také zařízení, v jejichž sítích je používán NAT (network address translation). To je bohužel limitující faktor, který sebou tyto bezpečnostní omezení přináší. Problémovou je pak situace, kdy za těmito zařízeními jsou oba komunikující uzly. V tomto případě je třeba, aby spojení zprostředkoval centrální server na internetu.

3.9 Transparentnost

V distribuovaných systémech byla obvykle transparentnost spojována se schopností průhledného připojení do lokálního systému. P2P software by neměl vyžadovat k tomu, aby mohl být používán, nějaká významná nastavení sítě nebo zařízení. P2P systémy by měli pracovat na internetu, intranetu, v privátních sítích a s vysokorychlostním či vytáčeným připojením. Měli by také být dostatečně transparentní v používání koncových zařízení, to znamená, že by měli být schopni pracovat s notebooky, PDA, mobilními telefony a podobně.

3.10 Odolnost proti chybám

Jedním z primárních cílů P2P systémů je vyhnout se centrálnímu původci selhání. I když mnoho P2P systémů už toho dosáhlo, přesto často čelí problémům spojeným se selháním systému (odpojení, nedostupnost) nebo chybám na straně samotných uzlů. Některé P2P systémy se snaží tyto problémy řešit pomocí speciálních uzlů, které ukládají dočasně některé soubory, až do doby, kdy se v případě výpadku uzel znovu objeví na síti. Odpojení může mít za následek nedostupnost zdrojů. To může nastat v případě, že zdroj je nedostupný z důvodu síťového selhání nebo proto, že hostitel je náhle v offline stavu. P2P systémy rozdělují zodpovědnost za správu systému kompletně mezi uživatele. Zatímco v modelu Klient–server nese tuto odpovědnost plně jen server.

3.11 Součinnost

Přesto, že již existuje mnoho P2P systémů, stále ještě není dosaženo jejich kooperace. Aby toho mohlo být dosaženo, je třeba splnit určitá kritéria.

A to jak určit, zda je systém kompatibilní, jak systém komunikuje, tedy například jaké používá protokoly, jak v systému probíhá výměna dat a jaké úlohy je v systému možné vykonat. Zajistit, že systémy budou kompatibilní na nejvyšší protokolové úrovni a zjistit na jaké úrovni je bezpečnost v systému. V P2P systémech bylo vyvinuto značné úsilí pro zlepšení jejich součinnosti. V prostředí P2P systémů vznikla iniciativa, která se pokusila shromáždit komunitu vývojářů P2P systémů, za účelem sepsání pravidel, která by vedla k lepšímu pochopení jejich prací a nápadů. Osloveni byli jak vývojáři ad-hoc systémů, tak ti z již zavedených sítí. Této pracovní skupině předcházela podobná snaha a jejím vyústěním bylo založení Grid fora [1].

4 Praktická část, tvorba vlastní P2P aplikace

V předchozích kapitolách byla definována podstata a vlastnosti P2P architektury. Pro upevnění si znalostí nabitých při teoretickém studiu P2P systémů, je vhodné si tyto znalosti prakticky vyzkoušet. V dalších částech práce bude představena má vlastní P2P aplikace, jejímž návrhem a implementací jsem se zabýval.

Aplikace byla vytvořena v programovacím jazyce Java. Jako vývojové prostředí byl použit Netbeans 6.1. Aplikace je založena na principu čisté P2P, komunikace tedy probíhá přímo mezi jednotlivými uzly, bez jakékoliv účasti centrálního zprostředkovatele. Aplikace nejen, že komunikuje mezi P2P uzly, ale díky navrženému protokolu je její podstatou synchronizace dokumentů, které stanice sdílejí. Ve stručnosti si tedy představme, jak aplikace funguje. Podrobně bude princip protokolu vysvětlen v další části práce.

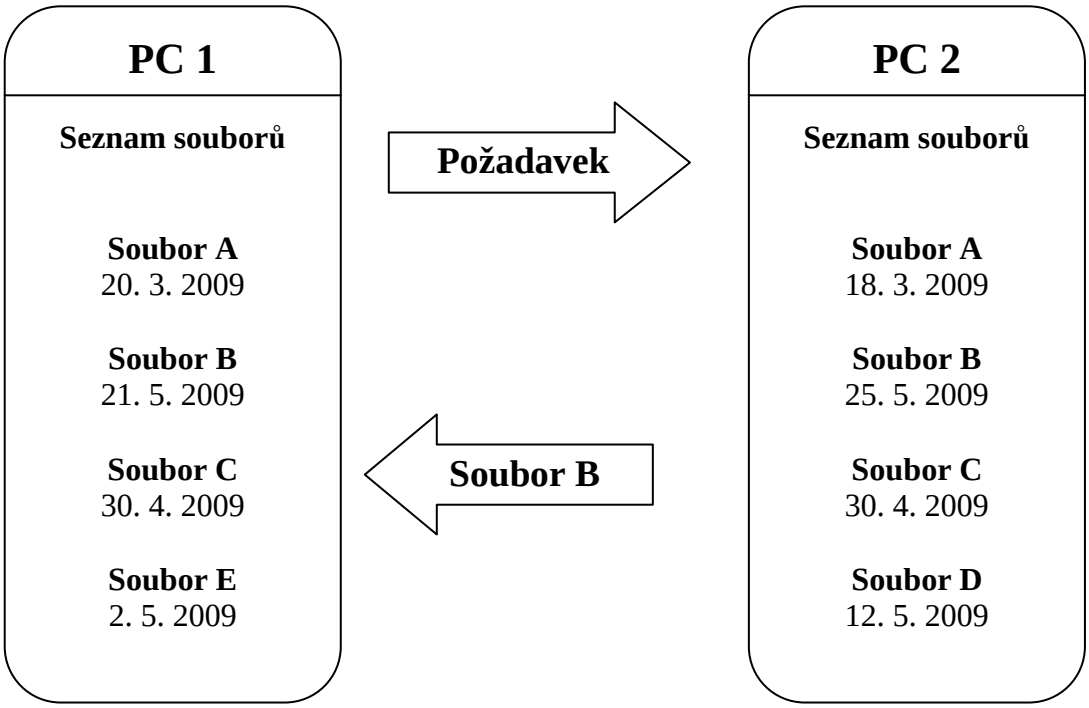
Předpokládejme, že máme určité dokumenty, například textové. Máme – li kolegu, se kterým spolupracujeme na projektu nebo nějakém úkolu, je třeba často kontrolovat průběžné výsledky práce, důležité pro další spolupráci. Toho lze dosáhnout například pravidelnými osobními konzultacemi, při kterých si dokumenty navzájem poskytneme. Jednodušším řešením je použití aplikace, jejímž návrhem jsem se zabýval.

K danému kolegovi se jednoduše připojíme a aplikace se postará o synchronizaci potřebných dokumentů. Synchronizace proběhne na základě znalosti názvů dokumentů a časů jejich poslední modifikace. Dalším kritériem, které by se případně dalo vyhodnotit, je shoda obsahu dokumentů. Kontrola obsahu je, ale již mimo zadání a téma práce.

Předpokladem je, že pokud spolupracujeme na stejném projektu, máme také dokumenty se stejnými názvy, proto je protokol aplikace navržen tak, aby pracoval jen s dokumenty, které vlastní obě stanice. Jak naše tak i kolegova. Pokud by aplikace byla navržena jinak, šlo by spíše o stahování souborů, takto je zdůrazněna podstata protokolu, kterou je samotná synchronizace dokumentů. V případě, že kolega vlastní dokument se stejným názvem a tento dokument aktualizoval, soubor se uloží na naší stanici do daného adresáře (obr. 2). Klíčovým faktorem při shodě názvů je tedy již zmíněný čas poslední úpravy daného souboru.

Aplikace může být využívána velmi dobře jako studijní pomůcka. Studenti, kteří pracují ve skupině na semestrální úloze a mají například od vyučujícího stejnou šablonu

se zadáním, kterou postupně upravují do finální podoby. Velmi snadno si mohou navzájem kontrolovat výsledky své práce. Dalším možným místem, kde by mohla aplikace najít své uplatnění jsou firmy. Spolupracovníci, kteří pracují na svých projektech si mohou stejně jako studenti snadno kontrolovat průběžné výsledky týmové práce na společných projektech.



Obrázek 2: Obecné schéma principu činnosti navrženého protokolu

5 Protokol a princip činnosti navržené P2P aplikace

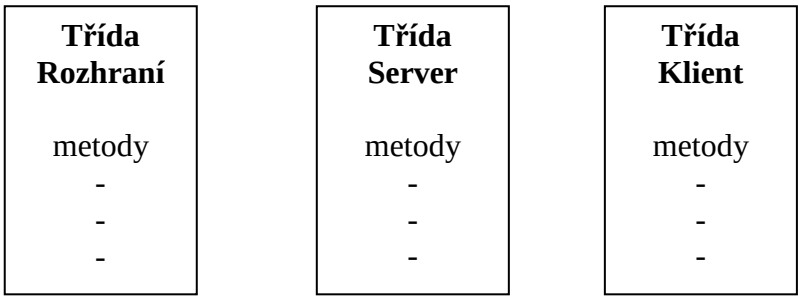
V této části práce bude podrobně rozebráno fungování navržené aplikace. Hlouběji budou vysvětleny její principy činnosti a budou popsány algoritmy, na kterých je založena. Pro přehlednost bude fungování aplikace ilustrováno na několika schématech. V některých zajímavých případech budou popsány i části zdrojového kódu aplikace.

Zdrojový kód aplikace, vytvořené ve vývojovém prostředí Netbeans, je rozdělen do *tříd*. Třída (Java class) je v objektovém programování základním stavebním kamenem. Představuje soubor proměnných, konstant a metod. Naše aplikace je konkrétně složena ze tří takovýchto tříd. Jsou jimi třída *Rozhraní*, třída *Klient* a třída *Server*. Každá třída obsahuje pro ní specifické metody (obr. 3). Třída *Rozhraní* obsahuje hlavní metodu aplikace Main a metody objektů grafického uživatelského rozhraní. Třída *Server* obsahuje metody pro příjem souborů, zpracování souborů, posílání souborů a další pro ni potřebné metody. Třída *Klient* stejně jako třída *Server* obsahuje metodu pro příjem souborů nebo vytvoření seznamu souborů. Obě třídy *Server* i *Klient* obsahují navíc metodu Run. Třídy *Server* a *Klient* jsou odvozeny od třídy *Thread*. Po zavolání metody Run běží třída Klient i třída Server každá v samostatném *vlákně*. Vlákna (threads) jsou samostatně běžící části programu, kterým je přidělován určitý čas běhu procesoru. Jednotlivé metody a jejich využití v protokolu budou představeny dále.

Po spuštění aplikace může uživatel jednak poskytovat své soubory ostatním a zároveň od ostatních synchronizované soubory požadovat. Tyto činnosti jenž nám aplikace umožňuje, zajišťují již zmíněné třídy *Klient* a *Server*. Nyní si vysvětlíme, jak aplikace realizuje samotné spojení, synchronizaci a kopírování souborů.

Spuštěná aplikace sice běží, ale sama o sobě ještě nic nevykonává a je zatím nečinná. Uživatel si sám s pomocí grafického uživatelského rozhraní zvolí jak bude aplikace dále fungovat. Pokud je aplikace uvedena stisknutím tlačítka do stavu online, zavolá se metoda run ve třídě *Server*. Aplikace naslouchá na daném portu a pouze čeká na příchozí spojení. V případě, že se náš kolega nebo někdo jiný rozhodne s námi synchronizovat dokumenty, opět pomocí interakce s grafickým uživatelským rozhraním zahájí proces spojení s naší aplikací.

Aplikace pro synchronizování dokumentů V P2P síti



obrázek 3: Fyzická podoba projektu aplikace ve vývojovém prostředí Netbeans

Nejprve je zavolána metoda zajišťující načtení seznamu souborů, metoda je volána ze třídy *Klient*. Ta má v proměnné uloženu cestu k adresáři, který obsahuje soubory poskytnuté pro případnou synchronizaci. Metoda načte do pole řetězců obsah adresáře, přesněji dva důležité parametry souborů v adresáři. Jsou to názvy souborů a časy poslední modifikace souborů. Parametry jsou od sebe odděleny znakem (;). Čímž nám vznikl seznam souborů ve formátu *CSV*. Soubor ve formátu *CSV* je tvořen řádky, ve kterých jsou jednotlivé položky odděleny znakem čárka (,). V našem případě je místo čárky použit středník.

Část zdrojového kódu metody pro načtení seznamu souborů:

```
for (int i = 0; i < listOfFiles.length; i++) {
if (listOfFiles[i].isFile()) {
Parametry = ((listOfFiles[i].getName()) + ";" + listOfFiles[i].lastModified() + " ");
SeznamSouboru[i] = Parametry;
}
}
```

Pro přehlednost je následně zavolána metoda, která vytvoří textový seznam souborů. Uživatel pak má k dispozici ucelený soupis souborů, které poskytuje v textovém dokumentu a to včetně jejich parametrů. Metoda tedy vytvoří v aktuálním adresáři nový textový soubor s názvem *FileList*. Do tohoto souboru uloží obsah pole seznamu souborů, vytvořeného předchozí metodou pro načtení seznamu souborů.

V tuto chvíli jsou všechny metody zajišťující lokálně na straně kolegy zpracování souborů vykonány a kolegův seznam souborů je připraven k odeslání. K tomu je nutné připojit se k naší aplikaci, která naslouchá na známém portu a IP adrese. Spojení je realizováno v metodě run třídy *Klient*. Pokud proběhne spojení úspěšně a seznam souborů v podobě *CSV* souboru je odeslán, uživatel je informován o vykonání operace. V případě, že se nelze spojit na známý port a IP adresu, tedy pokud je cílový kolega například ve stavu offline, je uživatel opět informován o nerealizovaném spojení. A tento proces je nutné zopakovat.

Po úspěšném odeslání seznamu souborů, kolegova aplikace vyčkává jaký bude výsledek zpracování a vyhodnocení seznamu souborů. Pozornost se tedy obrací na metody třídy Server na straně naší aplikace, které tyto potřebné činnosti zajistí.

Naše aplikace naslouchající na daném portu, akceptuje pokus o spojení a uživatel je o této akci informován. V případě, že uživatel bude chtít s daným kolegou synchronizovat soubory, pomocí interakce s grafickým rozhraním spustí mechanismus příjmu a zpracování souborů. Následně je zavolána metoda pro příjem souborů ze třídy Server. Metoda pro příjem zajistí načtení CSV souboru se seznamem souborů kolegy do proměnné typu String. O úspěšném přijetí seznamu souborů je uživatel aplikací informován.

Část zdrojového kódu metody pro příjem seznamu souborů:

```
BufferedReader cteni = new BufferedReader(new
InputStreamReader(soket.getInputStream()));
CsvKolega = cteni.readLine();
cteni.close();
```

Stejně jako v předchozím případě u kolegy, jsou i na naší straně aplikace následně zavolány metody pro načtení seznamu souborů a vytvoření textového dokumentu obsahujícího názvy souborů a časy jejich poslední modifikace. Metoda pro načtení souborů ze třídy Server, uloží navíc oproti metodě ze třídy Klient do pole řetězců absolutní cesty k souborům. Toto je spolu s tím, že metody jsou volány ze třídy Server jediný rozdíl oproti metodám volaným ze třídy Klient. Tyto metody již byly popsány výše, proto jejich funkčnost nebude v tuto chvíli dále rozebírána.

Seznamy souborů jak kolegův tak náš, jsou s pomocí dosud zavolaných metod uloženy v proměnných. Abychom mohli zjistit, které soubory jsou shodné, neshodné a které budou synchronizovány, je nutné seznamy souborů nejprve rozčlenit. K tomu je třeba oddělit od sebe názvy a časy poslední modifikace souborů. Teprve poté je možné testovat shodu názvů a následně testovat shodu časů poslední modifikace. V této části aplikace se skrývá jádro protokolu, bez nějž by synchronizace dokumentů nebyla možná.

Samotné rozdělení našeho a příchozího seznamu souborů zajišťují čtyři metody obr. 4. A to metoda pro získání našich lokálních názvů souborů a metoda pro získání našich lokálních časů posledních změn souborů. Dále pak obdobné metody pro získání názvů souborů z příchozího *CSV* souboru a jejich časy poslední modifikace. Oba seznamy souborů jsou ve formátu *CSV* odděleny středníkem..

Po vykonání těchto metod, vzniknou čtyři nová pole řetězců. Pole s našimi lokálními názvy souborů a s časem změny našich lokálních souborů. Dále pak pole s názvy z příchozího seznamu souborů a pole obsahující čas poslední úpravy příchozích souborů. S těmito čtyřmi poli, bude dále protokol pracovat a především na jejich základě bude rozhodnuto které soubory budou synchronizovány a které nikoli.

Část zdrojového kódu metody pro získání časů poslední modifikace souborů:

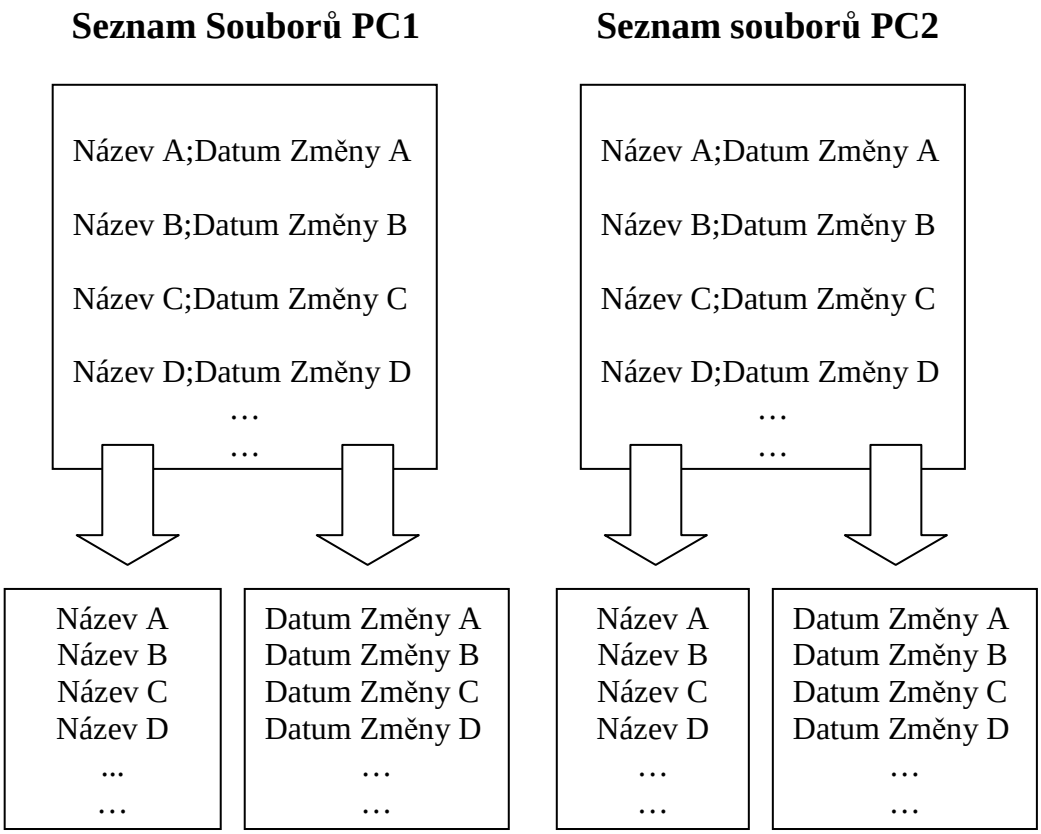
```
for (int i=0; i < SeznamSouborů.length; i++) {
    Scanner skener = new Scanner(SeznamSouborů[i]);
    skener.useDelimiter(";");
    String pom = skener.next();
    skener.useDelimiter(";");
```

```
if (skener.hasNext()) {
PoleDatumZmeny[i] = skener.next();
}
skener.close();
}
 }
```

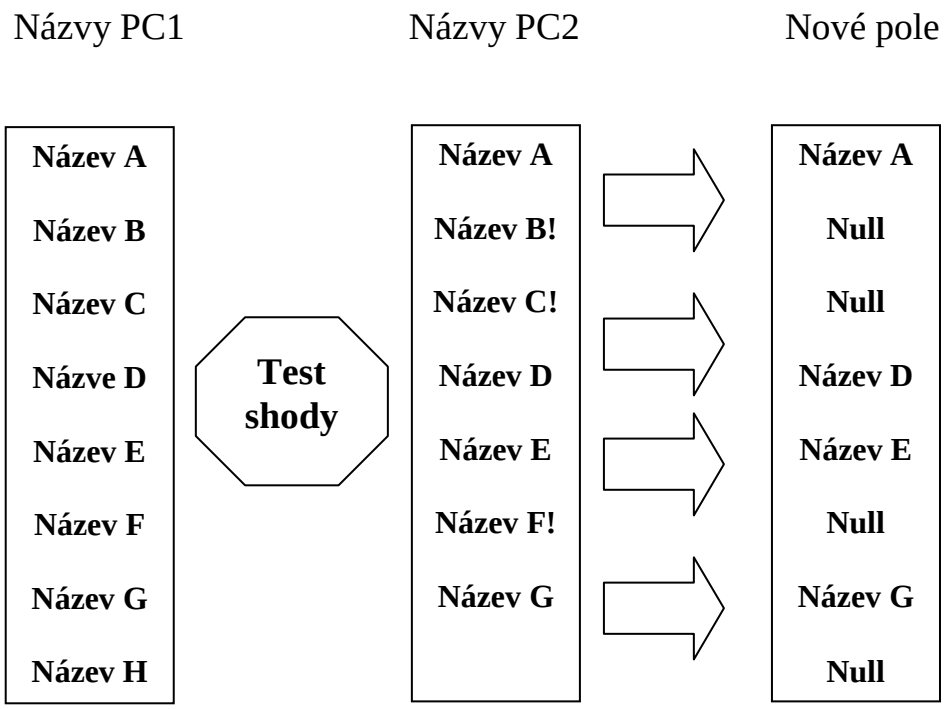
V tuto chvíli jsou všechna důležitá data, potřebná k dalšímu testování rozdělena do příslušných polí. Protokol nejprve zajistí testování na shodu názvů souborů, zavoláním příslušné metody, určené pro vykonání této činnosti (obr. 5). Metoda testuje v cyklu, postupně pro všechny prvky pole s našimi názvy souborů shodu s prvky pole s příchozími názvy souborů. Pokud nastane shoda názvů souborů, do nového pole jsou zapsány tyto shodné názvy. Pokud jsou názvy rozdílné pod daným indexem je pouze hodnota null. Tímto vzniklo nové pole, které obsahuje pod danými indexy názvy souborů nebo hodnoty null. Nyní je třeba zajistit zpracování tohoto nového pole.

Po prvním testování na shodu názvů bylo zjištěno, které prvky pole názvů kritérium shody splňují. Aby bylo možné dále pracovat pouze s těmito prvky, zavoláním metody pro zpracování nově vzniklého pole, budou odděleny od hodnot null (obr. 6). Pro délku tohoto pole je testováno, zda se prvek pod daným indexem rovná hodnotě null. Pokud podmínka není splněna, je index prvku, který obsahuje hodnotu přidán do ArrayListu. Použití ArrayListu má v této situaci své opodstatnění. Zajišťuje totiž potřebnou dynamičnost. Prvky v ArrayListu jsou tedy indexy našich názvů souborů, které jsou shodné s názvy souborů příchozích.

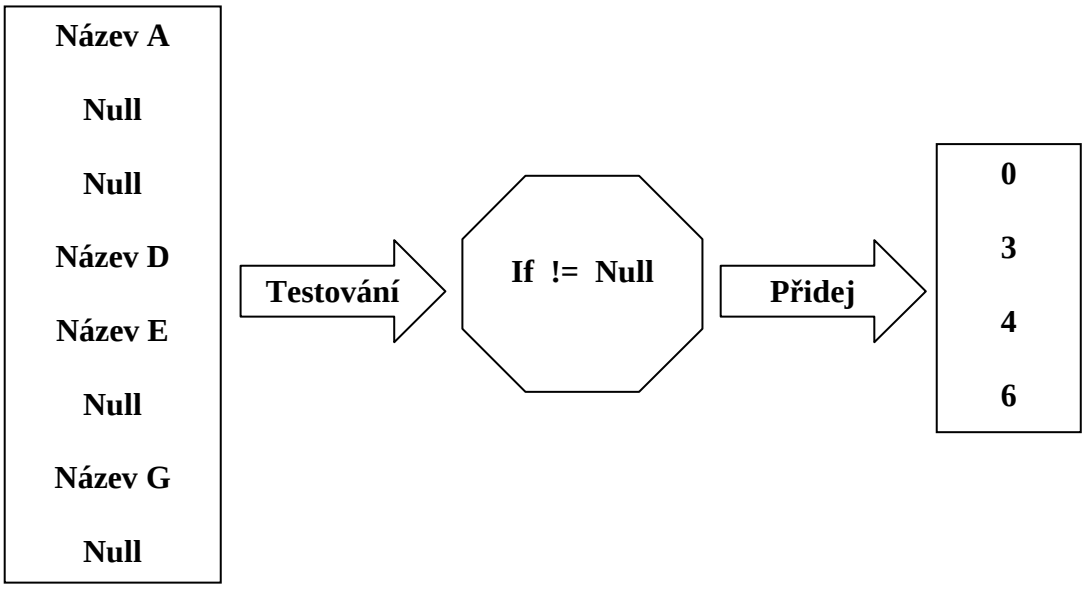
Metoda dále indexy shodných názvů převede z ArrayListu do nového pole. Pole bude obsahovat už pouze indexy potencionálních kandidátů na synchronizaci, vzniklých po prvním testování. Již zmíněná dynamičnost, je potřebná právě při vytváření tohoto pole. Pole má dynamickou délku, shodnou s velikostí ArrayListu.



Obrázek 4: Znázornění funkčností metod pro rozčlenění seznamu souborů



Obrázek 5: Testování na shodu názvů souborů



Obrázek 6: Zpracování nově vzniklého pole a získání indexů kandidátů po prvním testování

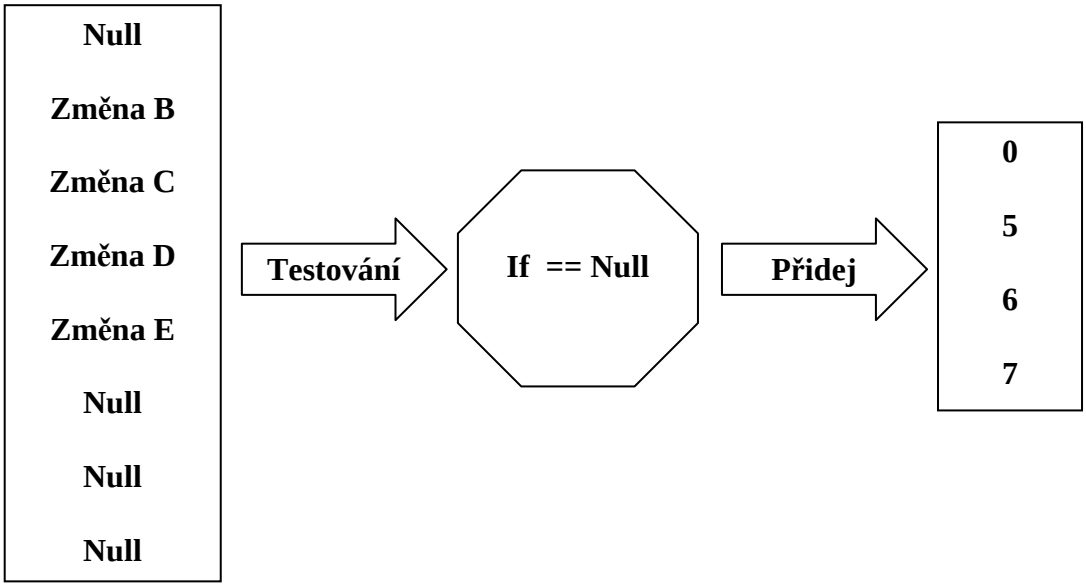
Po testu a zpracování testovaných názvů, jsou získány indexy souborů které splnily první kritérium shody názvů. Tyto soubory jsou kandidáty pro synchronizaci a bude s nimi dále pracováno. U souborů, které tuto podmínku nesplnily, je jasné, že pro synchronizaci vybrány nebudou. Soubory je ještě třeba otestovat na shodu časů jejich poslední modifikace.

Metoda pro porovnání časů poslední změny souborů, pracuje podobně jako metoda pro porovnání shody názvů souborů. V cyklu, pro délku pole s našimi údaji o poslední změně souboru, provede porovnání s polem s časy změn souborů příchozích. Tímto testováním opět vznikne nové pole. Toto pole bude taktéž obsahovat danou hodnotu při shodě časů posledních změn. V opačném případě hodnotu null. Metody pro testování shody názvů a časů posledních změn souborů, jsou tedy prakticky totožné. Proces zpracování se však od procesu zpracování shodných názvů podstatně liší.

Touto podstatnou změnou je fakt, že v tomto případě je hodnota null žádoucí. Druhým kritériem, který musí soubor splnit, jak již bylo vysvětleno dříve, je nutnost rozdílu v čase poslední úpravy. A pouze tehdy, jestliže při testu shody byla do nového pole pod daným indexem přidána hodnota null, je splněna tato druhá podmínka nutná pro synchronizaci (obr. 7). Metoda pro zpracování stejných změn úprav souborů, zajišťuje dynamičnost opět použitím ArrayListu. Do tohoto ArrayListu jou přidány pouze indexy prvků, u kterých se vyskytla hodnota null. ArayList je následně převeden na pole řetězců. Toto pole uchovává indexy prvků, u kterých nebyla zjištěna shodnost časů poslední úpravy. Po získání kandidátů se shodnými názvy, máme tedy již i kandidáty pro synchronizaci s rozdílnými časy změn poslední úpravy. Protokol nyní zajistí porovnání těchto polí.

Část zdrojového kódu metody pro zpracování shodných časů změn souborů:

```
for (int i=0; i < StejneZmeny.length ; i++) {
  if (StejneZmeny[i] == null) {
  arayZmeny.add(i);
  }
  }
```



Obrázek 7: Zpracování nově vzniklého pole a získání indexů kandidátů po druhém testování

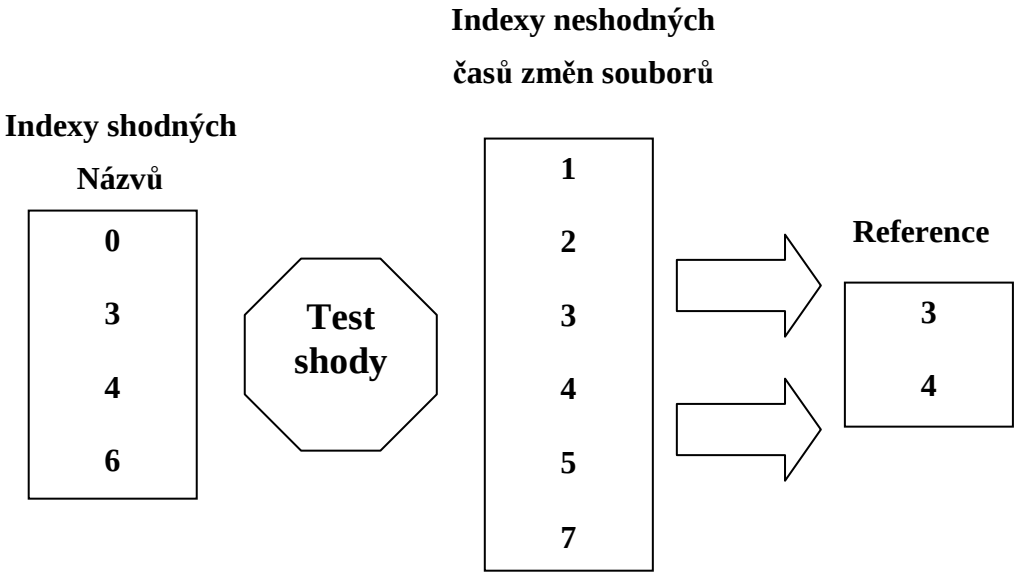
Obě nově vzniklá pole indexů potencionálních kandidátů, kteří splnily jednu ze dvou podmínek, potřebných k synchronizaci budou zavoláním příslušné metody otestovány. A pouze ty prvky, které splňují obě podmínky, mohou být vybrány pro synchronizaci. Připomeňme si nejprve význam získaných polí s indexy prvků.

Na počátku, aplikace zavolá metodu pro načtení seznamu souborů. Tyto soubory jsou v poli seřazeny postupně pod indexy od 0 do n. Po prvním testování na shodu názvů, získáme pole obsahující jednak shodné názvy a jednak hodnoty null. Algoritmus zajistí jen výběr prvků, které obsahují hodnotu. Aby se však na ně šlo později opět odkázat, do nového pole jsou uloženy pouze indexy těchto prvků. Tyto indexy odpovídají indexům prvků, v původním seznamu souborů. Tímto je zajištěno, že reference pod daným indexem odkazuje stále na tentýž soubor. Stejný význam mají samozřejmě i reference skrývající se pod indexy prvků, které byly testovány na shodu změny poslední modifikace souborů.

Samotné testování probíhá velmi snadno. Pro každý prvek pole s indexy shodných názvů je testována shoda s prvky pole s indexy rozdílných změn poslední modifikace (obr. 8.) V případě, že je testování úspěšné a v obou polích je reference na stejný soubor, je reference pod daným indexem přidána do ArrayListu. Následně je metodou pro zpracování zajištěno převedení ArrayListu do nového pole. V tomto poli máme tedy již reference na soubory, které mají stejný název se soubory příchozími a zároveň

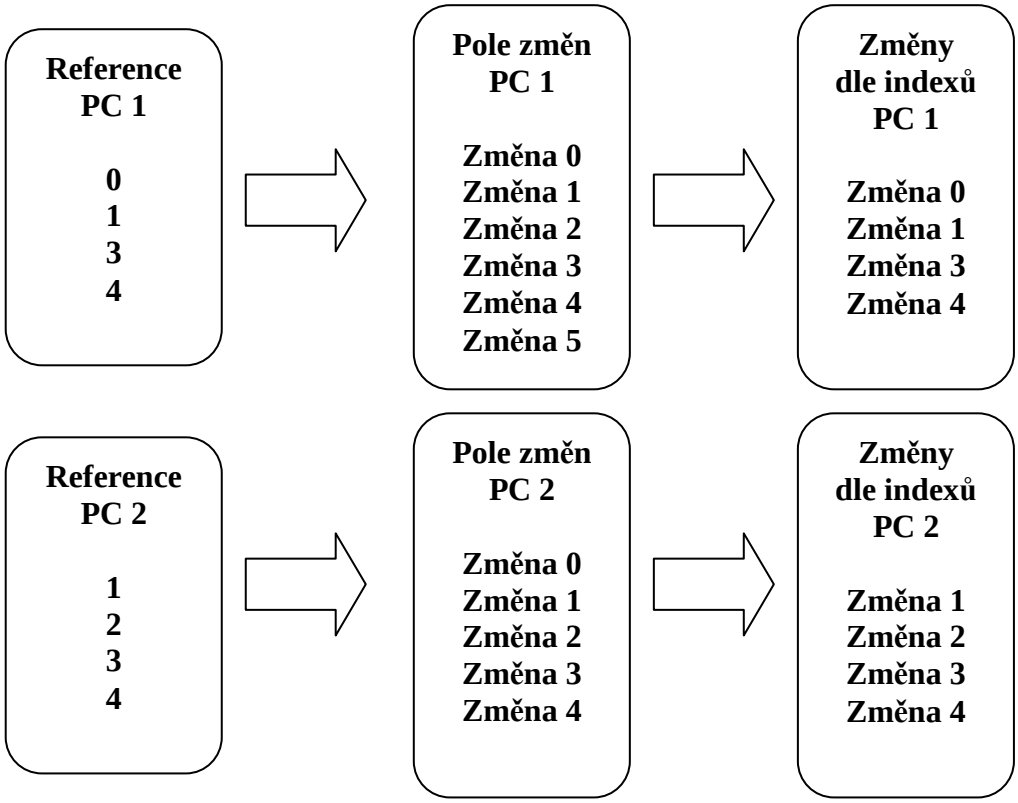
nejsou vytvořeny nebo upraveny ve stejnou chvíli. To znamená, že kritéria pro synchronizaci nejsou splněna v plné míře.

Připomeňme, že podstata navrženého protokolu je v synchronizaci souborů novějších. V této chvíli by tyto soubory kolegovi který chtěl s námi synchronizovat sice byly poslány. Zároveň v případě, že by jsme vlastnili soubor se stejným názvem, ale se starším datem úpravy než kolega, byl by také odeslán. Tento problém je třeba ještě dále ošetřit.



Obrázek 8: Získání referencí na soubory které splnily první dvě kritéria testování

Testováním jsme již získali reference na soubory, které se shodují v názvu a liší se časem úpravy. Nyní tyto soubory musí splnit poslední kritérium, aby byly vyhodnoceny jako vhodné pro synchronizaci. Metoda pro zpracování kandidátů synchronizace pracuje s časy poslední úpravy souborů (obr. 9). Testuje pouze ty soubory, které splnily předchozí dvě kritéria. Tyto soubory se skrývají pod příslušnými indexy v poli referencí. Soubory následně porovnává s časy změn souborů příchozích, na které odkazují taktéž reference pod danými indexy, a které rovněž splnily předchozí dvě kritéria.



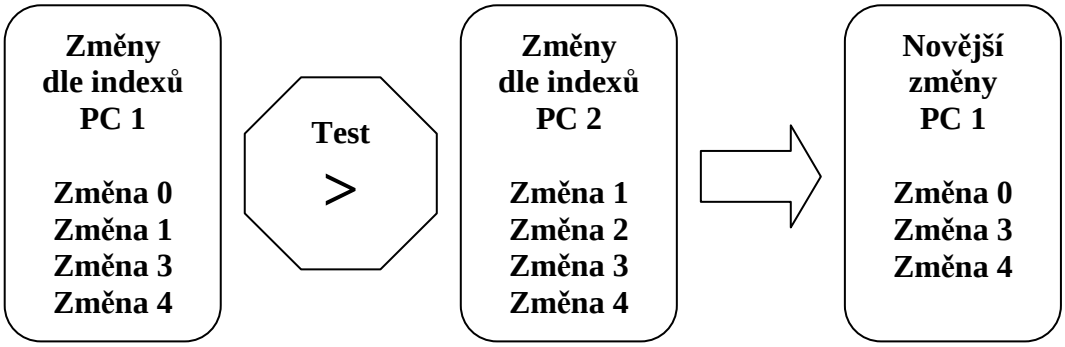
Obrázek 9: Indexy ukazující na prvky v poli změn, vytvoření nového pole z těchto prvků

Pokud je čas poslední úpravy našich souborů větší, to znamená byl upraven později než soubor příchozí. Soubor je vyhodnocen jako novější a čas jeho poslední úpravy je přidán do nového pole. Dynamičnost tohoto zpracování je opět zajištěna použitím ArrayListu. Teprve poté jsou prvky převedeny do tohoto nového pole. Toto pole obsahuje časy posledních úprav souborů, které máme skutečně novější nežli soubory které vlastní kolega (obr. 10).

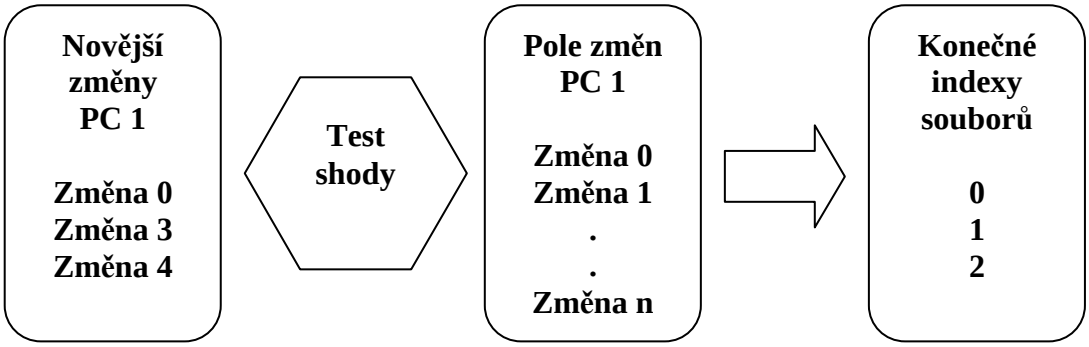
Posledním testem algoritmus vyřadil soubory, které nesplnily podmínky pro synchronizaci. Ztratili jsme ale reference pod danými indexy, které budou potřebné k posílání souborů. Snadno je však opět získáme. Porovnáme původní seznam posledních změn našich souborů se seznamem souborů které máme novější než kolega.

Hodnoty se musejí rovnat, vzešly právě z našeho původního seznamu časů posledních modifikací. Toto pole vzniklo již v úvodu, oddělením od názvů našich souborů. Algoritmus porovnáním nalezne, které prvky původního pole odpovídají prvkům pole které vzniklo testováním. Následně do pole vloží indexy těchto souborů.

V konečném poli máme finální reference na soubory (obr. 11). Tyto soubory mají stejný název jako soubory příchozí, jsou zároveň novější než soubory příchozí. Splňují tedy kritéria, která byla stanovena při návrhu protokolu pro synchronizaci dokumentů.



Obrázek 10: Získání novějších prvků po posledním testování



Obrázek 11: Získání konečných indexů souborů určených pro synchronizaci

Část zdrojového kódu metody pro zpracování kandidátů synchronizace:

```
for (int i = 0; i < KandidatiS.length; i++) {
    for (int j = 0; j < KandidatiK.length; j++) {
    }
    if (KandidatiS[i] > KandidatiK[i])
       Konecne.add(KandidatiS[i]);
}
```

```
PoleKonecne = new String[Konecne.size()];
    for (int i=0; i < PoleKonecne.length; i++) {
    PoleKonecne[i] =  Konecne.get(i).toString();
     }

    for (int i = 0; i < DatumZmenyS.length; i++){
      for (int j = 0; j < PoleKonecne.length; j++) {
        if (DatumZmenyS[i].equals(PoleKonecne[j]))
            JenVetsi.add(i);
       }
      }
```

Testování je ukončeno. Soubory, které budou synchronizovány jsou vybrány. Aby mohly být odeslány kolegovi je třeba zajistit ještě několik dalších věcí. Tou první je výběr absolutních cest k souborům. Již při volání metody pro načítání souborů bylo vytvořeno pole absolutních cest k souborům. Toto pole má jako všechny ostatní, své prvky indexovány. Metoda pro výběr absolutních cest použije pole s indexy konečných prvků a právě pod těmito indexy vybere absolutní cesty k synchronizovaným souborům. Vznikne tedy pole, kde jsou absolutní cesty k synchronizovaným souborům.

Dále je třeba získat názvy synchronizovaných souborů. Příslušná metoda využije stejného principu jako metoda pro získání absolutních cest. Z již dříve vytvořeného pole názvů souborů, vybere ty soubory, které se shodují s polem konečných indexů.

Ještě než dojde k samotnému poslání souborů zpět kolegovi, je zavolána metoda pro načtení obsahů souborů. Metoda uloží do pole řetězců, obsahy jednotlivých souborů. Obsahy načte s využitím pole absolutních cest k synchronizovaným souborům. Pole obsahuje právě tolik prvků, kolik je synchronizovaných souborů. Obsahy souborů, jsou tak pod stejnými indexy jako názvy synchronizovaných souborů. Ve chvíli, kdy jsou názvy i obsahy synchronizovaných souborů v příslušných polích, je vše připraveno k odeslání souborů kolegovi.

Kolega ve chvíli kdy nám odeslal svůj seznam souborů, pouze čekal na naši odpověď. Po odeslání ukončil komunikaci na daném portu. Zároveň na novém portu začal naslouchat a čekat na příchozí synchronizované soubory.

Naše strana aplikace zná port na kterém kolega naslouchá a pokusí se k němu připojit. Pokud je spojení úspěšné, kolegovi je odesláno nejprve pole s názvy novějších souborů (obr. 12). Za každý prvek pole je navíc přidán znak středník (;). Kolega přijme spojení a příchozí soubory jsou uloženy do proměnné. Bezprostředně poté je zavolána metoda pro zpracování názvů synchronizovaných souborů.

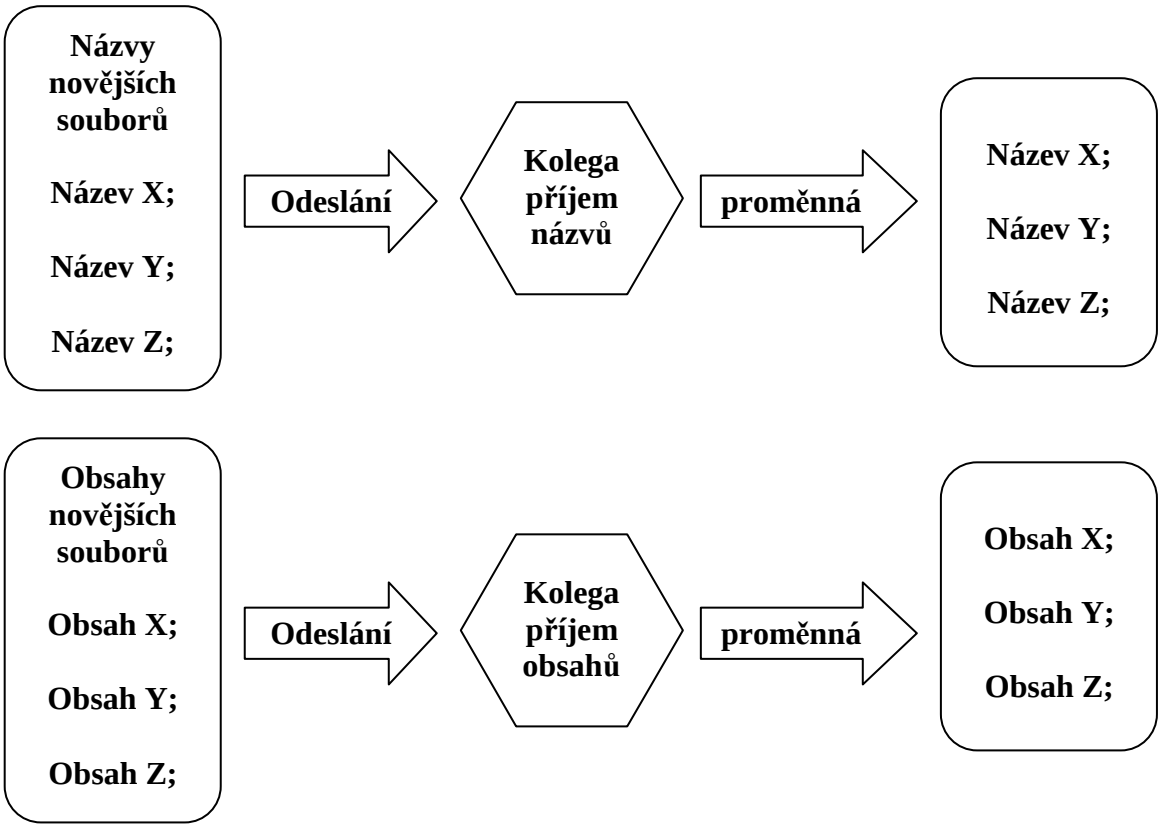
Metoda využívá znovu třídy Java.util.Scanner. Pracuje podobně jako metoda pro příjem seznamu souborů, ze třídy Server. V novém poli máme tedy názvy všech příchozích souborů.

Kolega obdržel sice názvy souborů, potřebuje ještě získat jejich obsahy. Proto znovu zahájí naslouchání na novém portu a čeká na spojení.

Ze třídy Server naší aplikace je zavolána metoda pro poslání obsahů novějších souborů. Metoda zajistí spojení na známy port a následné odeslání pole s obsahy novějších souborů, kolegovi. Za každý prvek pole s obsahy souborů, je opět přidán oddělovač středník. Ve chvíli kdy kolega spojení přijme, činnost na naší straně aplikace je ukončena. Všechny potřebné metody byly úspěšně vykonány a synchronizace dokumentů se zdařila.

Kolega nyní příchozí obsahy souborů uloží znovu do nové proměnné. Obsahy souborů jsou uloženy do nového pole. Pole s názvy synchronizovaných souborů má stejnou délku jako pole s obsahy souborů. To znamená, že soubor pod indexem 1 v poli názvů má svůj obsah taktéž pod indexem 1 v poli obsahů.

Zbývá tedy vytvořit v novém adresáři nové soubory, jejichž jména obsahuje pole názvů. Tyto nově vytvořené soubory budou naplněny polem s obsahy souborů. Zavoláním metody, která toto vykoná, je synchronizování dokumentů úspěšně dokončeno. Výsledkem jsou kopie novějších souborů, které má kolega uloženy v příslušném adresáři na svém pevném disku.



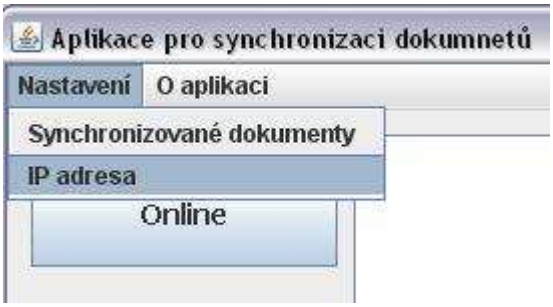
Obrázek 12: Odeslání novějších souborů kolegovi, příjem a následné uložení do proměnných

Část zdrojového kódu metody pro vytvoření kopií souborů:

```
for (int j=0; j < SynchNazvy.length; j++) {
    BufferedWriter out = new BufferedWriter(new FileWriter(Down + "/" + SynchNazvy[j]));
    out.write(SynchObsahy[j]);
    out.close();
    }
```

6 Grafické uživatelské rozhranní

Obrázek 13: Aplikace po spuštění



Obrázek 14: Položky v menu Nastavení

Grafické uživatelské rozhranní (obr. 13) se skládá ze tří tlačítek pro obsluhu aplikace. Dále z menu Nastavení a menu O aplikaci. Nechybí textová oblast, kde aplikace informuje uživatele o své činnosti. Před zahájením spojení je nutné zadat IP adresu, na kterou se aplikaci připojí. Druhým úkonem je vybrání adresáře, do kterého se uloží nové synchronizované soubory.

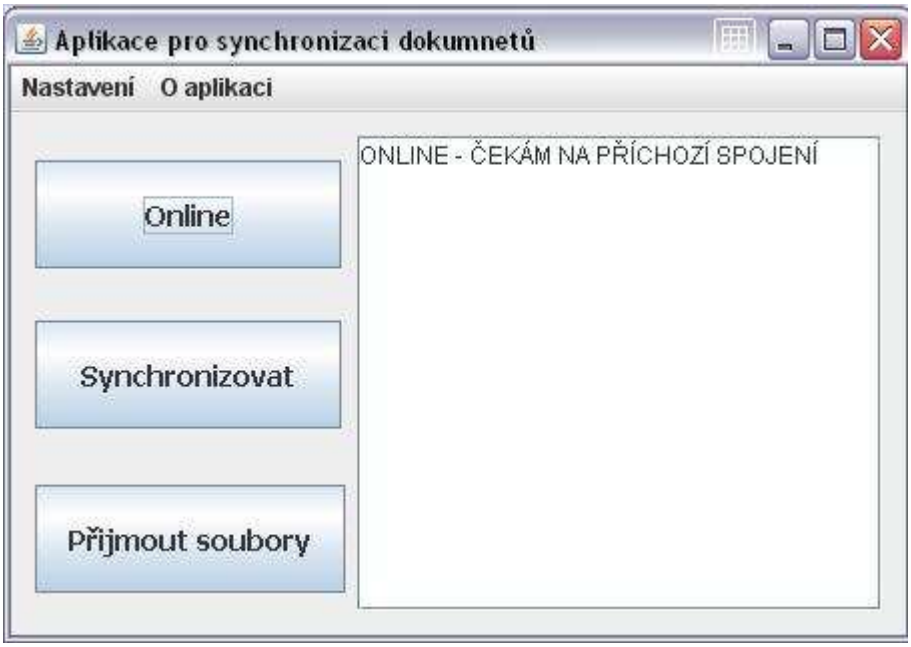
Obrázek 15: Dialogové okno po stisku tlačítka Synchronizované soubory v menu Nastavení

Pokud se uživatel rozhodne synchronizovat dokumenty, je nutné nastavit základní parametry programu. Rozbalit menu Nastavení (obr. 14), které obsahuje dvě položky. Po stisku tlačítka Synchronizované soubory, je uživatel pomocí dialogového okna vyzván k vybrání úložiště, pro nově příchozí soubory (obr. 15). Po stisku tlačítka IP adresa, je uživatel požádán o zadání IP adresy (obr.16), ke které se chce připojit.

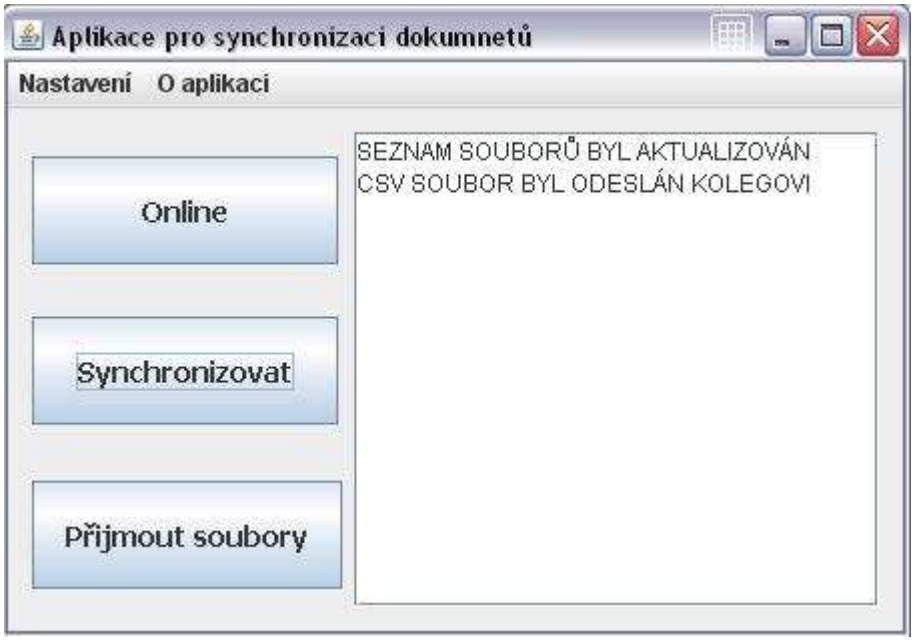
Obrázek 16: Dialogové okno po stisknutí tlačítka IP adresa v menu Nastavení

Po základním nastavení programu uživatel pouze stiskne tlačítko Synchronizovat (obr. 18). Vše ostatní vykoná aplikace na pozadí. Uživatel je o výsledku proběhlé akce, informován výpisem do textové oblasti.

Pokud uživatel s nikým konkrétně synchronizovat soubory nechce, může být pouze ve stavu Online (obr. 17). V tom případě není nutné program nastavovat. Po stisku tlačítka Online, je uživatel k dispozici ostatním. Kolega, který zná IP adresu uživatele, se k němu může jednoduše připojit a synchronizovat.



Obrázek 17: Aplikace po stisku tlačítka Online



Obrázek 18: Aplikace po stisku tlačítka Synchronizovat

7 Závěr

Tato práce se zabývala studiem počítačových systémů, založených na peer to peer architektuře. Byly zde uvedeny typické prvky, algoritmy a terminologie, které jsou v P2P systémech používány.

P2P systémy nejsou řešením pro všechny problémy, které se v budoucnu v oblasti počítačových systémů vyskytnou či se vyskytují. Jsou však velmi dobrou alternativou k tradičním systémům a architekturám. Těmi jsou centralizované systémy a architektura Klient-server. P2P systémy jsou vhodným kandidátem pro odstranění omezení v oblasti rozšiřitelnosti, anonymity a odolností proti chybám.

Řešení praktické části poskytlo dobrou příležitost, ověřit si nasazení P2P aplikací v praxi. Vznikla plně funkční aplikace, která může sloužit jako studijní pomůcka pro studenty. Stejně jako studenti, ji mohou využívat i kolegové v různých oborech své činnosti. Pokud mají potřebu pravidelně synchronizovat své dokumenty s ostatními.

8 Literatura

[1] MUTHUSAMY, V. : An introduction to peer-to-peer networks. Presentation for MIE456 - Information Systems Infrastructure II, Miami 2003.

[2] MILOJICICI, D. ,KALOGERAKI, V. , LUKOSE, R. , NAGARAJA, K., PRUYNE, J., BRUNO, R., ROLLINS, S.: Peer-to-peer computing, University of California 2002

[3] STROULIA, E.: P2P Architecture, University of Alberta 2006

[4] RAMANATHAN, M. K., KALOGERAKI, V. PRUYNE, J.: Cindiny Good Peers in the Peer to-Peer Networks. International Parallel and Distributed Computing Symposium, Fort Lauderdale, Florida, April 2002.

[5] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable contentaddressable network. In *Proceedings of the ACM SIGCOMM 01 Conference*, San Diego, CA, Aug 2001.

[6] FRIEDMAN, A.: Peer-to-peer security, Hardvard University 2005

[7] SUBRAMANIAN R., GOODMAN B.D., :Peer-to-peer Computing: The Evolution of a Disruptive Technology, Idea Group Inc 2005, ISBN 1591404304

[8] TAYLOR I.J.,:From P2P to Web Services and Grids: Peers in a Client/Server World, Springer-Verlag, London 2004, ISBN 1852338695

[9] MILLER M.,: Discovering P2P, 2001, ISBN-10: 0782140181, ISBN-13: 978-0782140187

[10] Sun Microsystems, Inc., The Java Tutorials, volně ke stažení z http://java.sun.com/docs/books/tutorial/